Software Estimation: Perfect Practice Makes Perfect

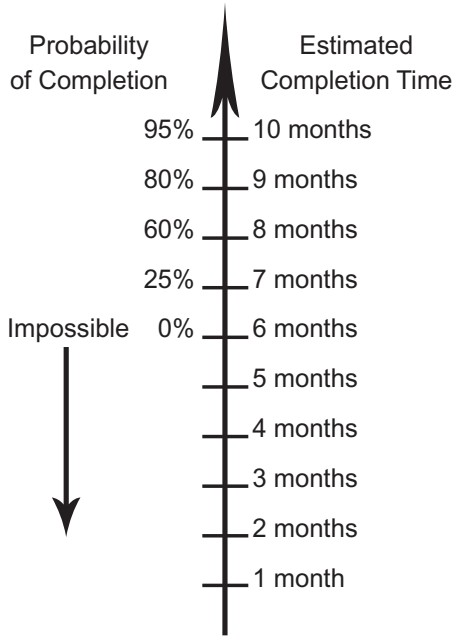
David Henry
Linux NetworX

Accurate software estimation has long been a headache for software developers. Much of the problem stems from lack of estimation training and practice. This article is the result of the author's in-the-trenches experimentation with different estimation training methods and is intended to give practical advice to managers or technical leads who wish to initiate a software estimation process in their organization

Software estimation is a difficult problem. Some have called it a *black art*. How does one go about learning this black art? Actually, it is not as exciting as it sounds: It takes practice and hard work. Unless developers get regular practice doing estimation work, improvement will be difficult. My son's karate instructor shouts his philosophy about practice at the end of each karate session: "Practice doesn't make perfect! Perfect practice makes perfect!"

Most software developers do some estimation work, but many are not trained to do it properly. Regular estimation work with feedback gives the developer the opportunity to improve his/her estimation skills. This article presents a few ideas about how to involve software developers in the estimation process. The techniques presented are being used at the author's organization with some success; result data will be presented later in the article.

Figure 1: *Estimates Are Probability Statements*



Why Software Projects Are Poorly Estimated

There are many misconceptions among software developers about software estimation, which leads them to create poor estimates. The first thing a software developer should understand is *what* an estimate really is; estimates are probability statements. For example, if a developer believes that he/she has an 80 percent chance of completing a project in nine months (see Figure 1), but his/her manager says it has to be done in seven months, what just happened?

By moving the completion date arbitrarily, the manager has just reduced the probability of on-time completion to 25 percent (assuming no other parameters are changed). This was probably not the manager's intention. Most people would probably feel very uncomfortable if they knew the project they were responsible for only had a 25 percent success probability. They would most likely make a vigorous effort to move the completion date back. By understanding and teaching the concept of estimates as probabilities, engineers can make a better defense of their estimates.

Software engineers should also understand the difference between *target setting* and *estimation*. Target setting is when a completion target is set because of some external dependency, such as a conference or fiscal year, and the engineer has to figure out how to meet that target. True estimation must be based on an analytical foundation with an estimation process that is not open to debate. The estimation process should be a black box with inputs of requirements, resources, etc., which generate the estimate (see Figure 2).

Inputs are *independent* variables and the output is a *dependent* variable. For example, if the manager wants to shorten the schedule, he/she must experiment with different inputs: adding more resources to the project or reducing the functionality of the end product. A defined estimation process is critical to any organization that desires repeatable, consistent, and quality estimations.

Another common problem is that the estimation task itself is often not scheduled as part of the project. This leads to what is known as *off-the-cuff* or *best-guess* estimates (or euphemistically as *expert judgment*) that most developers at one time have done and later regretted. It can be difficult to justify the importance of estimation work when there is a lot of pressure to start working on project deliverables.

The estimation work at the outset of a project is really just the tip of the iceberg: Creating infrastructure and the culture to collect and analyze metrics and other estimation inputs throughout the project can take a lot of organizational discipline and work. It is easier to use personal memory of past projects than to gather and analyze historical data. However, creating estimates from personal memory alone is a proven cause of cost and schedule overruns [1].

Is a high level of estimation accuracy even possible early in the project? The answer is, "It depends" (more about this later). In a less mature software organization, there are usually too many unknowns at the beginning of a software project to estimate the following with high precision and accuracy: Which technology will be used? How long will it take to learn it? What if it doesn't work as advertised? The list goes on and on. The author has been involved in many projects where project milestones are all mapped out at the beginning of the project (before critical decisions have been made) with pinpoint precision. This false precision gives the impression that the estimates are also accurate.[1]

With so many unknowns, estimating software can be like peering into thick fog. One can see things that are near pretty clearly and estimate their distance.

Objects farther away can only be dimly seen or not seen at all. How accurate can distance be estimated when the traveler has no map and cannot see the goal, but is relying on memory only? Early estimates should be presented in a way that expresses this potential inaccuracy. For example, the author has attended meetings where managers presented all completion estimates to the day (even long-term estimates). Once when one of these estimates was being presented (which was still over a year away), the question was asked, "Do you want the project completed at 10 a.m. or 11 a.m.?" That manager got the message and now gives all estimates farther than three months away in quarter precision, e.g., first quarter of 2002. Treating early estimates for what they are takes a lot of unnecessary pressure off the developers. Of course, estimates can and should be revised as the project progresses, since the end can usually be more clearly seen the closer it gets.

Now to explain the answer given earlier, "It depends." There are some software projects that are estimated with high precision *and* accuracy. If an organization has a tuned estimation process, metrics gathering, risk management, and other necessary processes in place, highly accurate estimates are certainly possible. The fog of project unknowns will bother this type of organization less. Its trained engineers have created an organization-specific map of the software development terrain and can more easily navigate around roadblocks. There are other situations where high accuracy can be obtained even with a less mature organization: projects where the engineering team has high familiarity with the domain and small non-complex projects.

Where to Start

Before proceeding to educate developers about software estimation, it is instructive to find out how skilled at estimation they are. Some people have a knack for the basic skill of estimating, and others need more practice. Try taking Jon Bentley's estimation quiz [2].[2] When taking the quiz, instruct the engineers to fill in upper and lower bounds that give one an 80 percent chance of including the correct value. If the engineer scores poorly, this is a chance to remind him/her to read requirements more carefully. The whole point of taking an estimation quiz before getting down to the nuts and bolts of estimating is to establish the *before* state, so the engineer can track his/her improvement at basic estimation skills.

The logical place for the engineer to do estimation work is within the scope of the organization's estimation process. If no estimation process exists, get one in place first. Defining the estimation process is outside the scope of this paper, but some online resources are given later.

Create a Feedback Loop

Developers need to keep track of their own day-to-day estimates to estimate accurately in the small. Estimation in the small is a different problem than estimation in the large, but it is an easier problem that should be tackled first. Most engineers will not need to do much estimating in the large unless they are technical leads or managers, so this paper will not address its unique problems.

Recently we interviewed a candidate for a software development position, and asked him how he liked to be managed. He replied: "I write embedded code for devices. Here's how it works – a sensor reads a device, providing input to the program, which then modifies the output to the device. The sensor repeatedly gets new values from the device, which allows the program to guide the device until the sensor detects values in the correct range. This means that the device is operating at the proper level."

He was describing a feedback loop, which is how he liked to be managed (with the manager as the program and the employee as the device). When managing people, a self-directed feedback loop is even better, which mostly takes the manager out of the picture. We have implemented such an e*mbedded controller* in our organization.

We have created an estimation spreadsheet that contains estimates for tasks in the product requirements. (The spreadsheet is available at <www.burgoyne.com/~henryd/estimation>.) At the beginning of each week, the engineer enters estimates for the weeks' work and the actual amount of time the task from the previous week had taken. This is the essential idea, and it only takes a few minutes per week. The spreadsheets are checked into our source code repository, so anyone can see others' spreadsheets.

The spreadsheet contains some simple formulas to allow the engineer to see at a glance how close his/her estimates are to reality. Since it is a spreadsheet, it can also be easily modified to suit different needs, and charts or graphs could be added. The manager can see who is currently working on what and how well the engineer is doing at estimation.

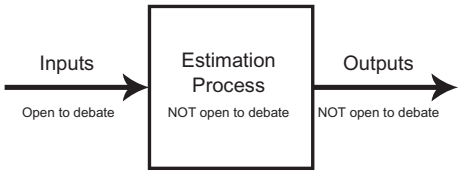


Figure 2: *Estimation Process as a Black Box*

There are certainly other ways of keeping track of what work is currently being done, but this solution is lightweight and does not require expensive or custom software. The main point is that the engineer should be teaching himself to estimate by practicing estimation on a regular basis and seeing the accuracy and results of those estimates.

There are a few caveats, however. As with any metrics gathering, it is best to explain at the outset that the gathered information will not be used *against* the employee (e.g., in reviews). Otherwise, engineers may fudge the numbers, rendering the data useless. We have found it useful to only enter time spent on the actual task, not including interruptions, personal time, etc. Because of this, the estimation spreadsheet does not differentiate between *actual* and *elapsed* task time. Elapsed time is the time between the start and finish points of a task (including interruptions and time spent on other tasks), whereas actual time is the time spent only on the task. We felt that elapsed time was more of a scheduling issue and did not belong in a tool used only for estimation purposes, although other organizations may see this issue differently. In any case, standardizing the metric helps simplify the process.

The tasks themselves should be small tasks of three days or less in size. This will force the developer to think about the details of the task, and the estimation should therefore be more accurate when broken into smaller pieces. For example, if a large task estimate (30 days) is underestimated by 50 percent, the estimation error will be larger than if the same task is broken into 10 smaller three-day tasks. It is unlikely that all 10 tasks will be underestimated by 50 percent: some will be underestimated and some overestimated, which will tend to cancel the estimation errors and produce a better overall estimate.

The developer should give three estimates: worst case (pessimistic), best case (optimistic), and most likely. When asked to give single-point estimates, many developers will simply provide best-case

estimates. Giving all three estimates will reveal such tendencies.

Our estimation spreadsheet has been in use for approximately eight months at the author's organization by engineers with little or no previous formal estimation practice. Before attempting any work-related estimation, all engineers involved in the project took the estimation quiz mentioned previously and scored poorly. During the first month of use, the average percent differences between estimates and actual task completion times were about 75 percent, and the variance in the group was quite large: 25 percent to 150 percent. By the third month of use, the average percent differences for the month had dropped to about 35 percent, with the greatest improvements among engineers who had never before given estimates in such detail. The average differences had stabilized to about 25 percent after six months, which may be the best average results to be obtained by this method.

Comments from the engineers suggest that the most useful feature of this method is the estimation history contained in the spreadsheet, which assists the engineer's memory when creating new estimates. Other comments suggest that this process helped the participants properly size lower- and upper-estimate bounds (shown as best- and worst-case estimates in the spreadsheet). This lightweight method seems to work best as an introduction to software estimation. If the organization wishes or needs better estimation accuracy, this method can be modified or discarded for a more rigorous method once the software team gains some estimating skill.

Use Group Estimation Processes

When making project estimates, get everyone's head into the game by using group estimation techniques. Wide band Delphi is a popular formal technique [3]. For those who prefer less formal methods, estimates can be made separately by members of the group and then averaged. This tends to have a conservative effect on the estimate. The wide band Delphi technique is good for groups with strong personalities because of its anonymous estimations. If strong personalities are not a problem, try reaching group consensus, but do not vote to overrule someone who may have a valid point.

We use another spreadsheet to aid our group estimation process. By first decomposing our project by requirement into modules/classes, each engineer separately estimates the size of each module in lines of code (LOC). In order to estimate size, we use estimation by analogy. For example, on a past project five new forms were to be added to the graphical user interface. By counting the LOC per form from the most recent project, it was fairly easy to compare new forms with previously built forms and estimate how many LOC the new forms would contain. After estimating separately we met, and our estimates were surprisingly similar. We discussed differences, which were mostly different assumptions about the requirements. We found this an effective way to create consensus that leads to a sense of *estimation buy-in* among the team.

Lessons Learned

The following lessons learned are suggested for best results:

- Do not rush estimates. Take the time to define and *use* a formal estimation process.
- Integrate regular estimation work into organizational processes.
- Use group estimation techniques, allowing team members to learn estimation techniques from one another.
- Gather estimation measurements and use them in future estimation efforts.
- Avoid *off-the-cuff* estimates. Managers should not be tempted to demand these types of estimates. Developers should resist giving an estimate until a detailed analysis of the problem has been made.

References

1. Lederer, A., and J. Prasad. "Nine Management Guidelines for Better Cost Estimation." Communications of the ACM Feb. 1992: 34-49.
2. Bently, Jon. Programming Pearls 2nd Ed., appendix 2. Addison-Wesley, 2000.
3. Wiegers, Carl. "Stop Promising Miracles." Software Development Feb. 2000.

Notes

1. For a good discussion of the difference between accuracy and precision, see Steve McConnell's "Rapid Development," Microsoft Press, 1996: 173.
2. The estimation quiz is available online at <www.cs.bell-labs.com/cm/cs/pearls/quiz.html>.

Free Estimation Software

- Construx Estimate 2.0. A user-friendly tool that combines several estimation methods. <www.construx.com/estimate>.
- COCOMO. Requires knowledge of COCOMO II. <http://sunset.usc.edu/research/cocomoii>.
- Cosmos. Requires knowledge of COCOMO II. <www-cs.etsu.edu/softeng>.
- SizeCost. A wizard-oriented tool for the beginner estimator. <http://members.tripod.com/~djelovic/sizecost.htm>.
- SoftEst. A full-featured COCOMO II implementation. <http://sepo.sparwar.navy.mil/estimation.htm>.

Author's Note

The author is not affiliated with any of the estimation tools noted above. There are of course many excellent estimation tools that can be purchased, but for an organization/developer just starting to work with estimation software, it may be easier to check out some of the simpler free tools first.

The estimation process should be tailored to fit the organization. There are some excellent estimation processes available publicly, including "Manager's Handbook for Software Development," Revision 1 (See section 3: Cost Estimating, Scheduling, and Staffing). It is available online at <http://sel.gsfc.nasa.gov/website/documents/docs/84-101.pdf>.

About the Author

David Henry is director of Software Engineering at Linux NetworX, a cluster computing company. He is also a principal of Synergy Software, a consulting firm, and occasionally teaches programming classes at Columbia College in Salt Lake. Henry has nine years of industry experience in software development. He has a bachelor's degree in computer science from Brigham Young University and is completing a master's degree in computer science at Colorado State University.

Linux NetworX
8689 South 700 West
Sandy, UT 84070
Phone: (801) 562-1010
Fax: (801) 568-1010
E-mail: dhenry@lnxi.com